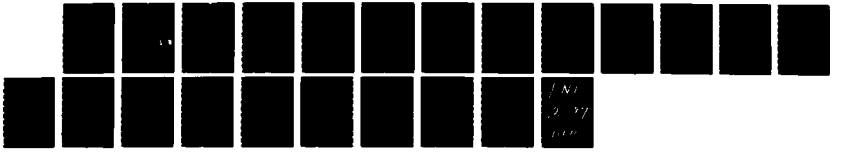
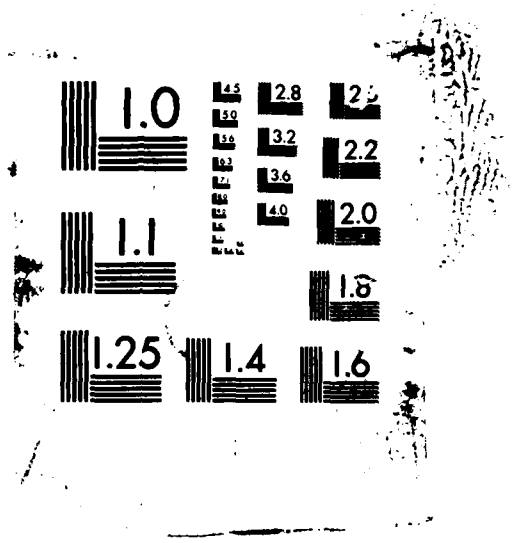
AD-A175 663 MULTILEVEL REPRESENTATION AND SIMULATION FOR VLSI DESIGN(U) CARNEGIE-MELLON UNIV PITTSBURGH PA DEPT OF ELECTRICAL AND COM S W DIRECTOR ET AL 26 NOV 86 1/1

UNCLASSIFIED ARO-20060 6-EL DAAG29-83-K-0083 F/G 9/5 NL




1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.8
3.2
3.6
4.0

MASTER COPY - FOR REPRODUCTION PURPOSES

(2)

AD-A175 663

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER: ARO 20060.6-EL	2. GOVT ACCESSION NO.: N/A	3. RECIPIENT'S CATALOG NUMBER: N/A
4. TITLE (and Subtitle): Multilevel Representation and Simulation for VLSI Design		5. TYPE OF REPORT & PERIOD COVERED: Final Report 5/23/83 - 5/22/86
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s): S. W. Director, D. E. Thomas		8. CONTRACT OR GRANT NUMBER(s): DAAG29-83-K-0083
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Carnegie Mellon University, Dept. of ECE, 5000 Forbes Avenue, Pittsburgh, PA 15213		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS: U. S. Army Research Office, Post Office Box 12211, Research Triangle Park, NC 27709		12. REPORT DATE: November 24, 1986
		13. NUMBER OF PAGES: 18
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)		15. SECURITY CLASS. (of this report): Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

DTIC ELECTE JAN 05 1987 S D D

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

NA

18. SUPPLEMENTARY NOTES

The view, opinions, and/or findings contained in this report are those of the author(s) and should not be construed as an official Department of the Army position, policy, or decision, unless so designated by other documentation.

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

VLSI design, Simulation, multilevel representation

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The design of integrated circuits has become almost impossible without circuit and system level simulation programs. Simulators at the circuit level produce performance data that allows the circuits to be functionally verified, optimized, and their performance evaluated. The size and complexity of integrated circuits has increased tremendously since the first circuit simulators were designed. Because the large circuits need to be simulated as thoroughly as smaller ones, the time spent simulating larger circuits has increased dramati-

DTIC FILE COPY

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

6 12 30 027

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

cally. Consequently, simulation tools that can efficiently simulate large circuits are needed. In this regard, work under this contract has focussed on the improvement of the mixed circuit/logic simulator SAMSON, as well as the development of new approaches to simulation. At the system level, simulators also produce data that allow the system architecture to be functionally verified, optimized and evaluated. Because of the ever larger systems being designed for implementation on an integrated circuit chip, there is a need for higher performance system level simulators. In this regard, work under this contract has also focussed on the development of new approaches to system level simulation. A study of the use of hardware accelerators in this application is described.

DTIC
ELECTE

Final Report, Contract DAAG29-83-K-0083

S. W. Director
D. E. Thomas
Electrical and Computer Engineering Department
Carnegie-Mellon University
Pittsburgh, PA 15213
26 November 1986

1. Introduction

The design of integrated circuits has become almost impossible without circuit and system level simulation programs. Simulators at the circuit level produce performance data that allows the circuits to be functionally verified, optimized, and their performance evaluated. The size and complexity of integrated circuits has increased tremendously since the first circuit simulators were designed. Because the large circuits need to be simulated as thoroughly as smaller ones, the time spent simulating larger circuits has increased dramatically. Consequently, simulation tools that can efficiently simulate large circuits are needed. In this regard, work under this contract has focussed on the improvement of the mixed circuit/logic simulator SAMSON, as well as the development of new approaches to simulation. At the system level, simulators also produce data that allow the system architecture to be functionally verified, optimized and evaluated. Because of the ever larger systems being designed for implementation on an integrated circuit chip, there is a need for higher performance system level simulators. In this regard, work under this contract has also focussed on the development of new approaches to system level simulation. A study of the use of hardware accelerators in this application is described.

2. SAMSON

Several years ago, the circuit simulator (SAMSON), was developed for the purpose of increasing simulation speed by using a "mixed level" strategy whereby critical parts of the circuit are simulated in a detailed manner (circuit level simulation), and non-critical parts of the circuit are simulated in a less detailed fashion (logic level simulation).

Under the current contract extensions to SAMSON were made. In particular mixed level circuit initialization was implemented and a basic change to the way SAMSON performs logic simulation was made. SAMSON needed mixed level circuit initialization because there was no easy way to specify the initial state of the entire circuit. The logic simulation algorithm was changed to yield more accurate modeling of circuit behavior and to allow the logic level simulator to handle bus interactions.

The improved version of SAMSON was tested by simulating a series of circuits to exercise its individual functions. The mixed level circuit initialization procedure consistently produced initial conditions and very rarely failed to find an initial stable circuit state. The logic simulator was shown to be reasonably

DTIC
COPY
INSPECTED

Dist | Special
A-1

fast when compared with similar programs, and quite accurate. The mixed level signal converters were shown to produce waveform conversions that are computationally efficient and do not excessively degrade the accuracy of the simulation. In trials with single level simulation, mixed level simulation produced comparable accuracy in the output waveforms while requiring much smaller runtimes.

The addition of impedance information to the logic level model enabled bus interactions to be simulated at the logic level. An empirical method was developed for determining logic level gate model delay parameters. Tests on the delay parameters produced by these methods showed the parameters to be very accurate in modeling the delays through circuit level gates.

3. CINNAMON: Coupled Integration and Nodal Analysis of MOS Networks

A novel approach for circuit simulation has been developed that promises a significant improvement over conventional methods. The algorithm involves an explicit event driven technique that seems stable even when the accuracy of the solution is relaxed, and is able to perform automatic and dynamic partitioning of the network, thus allowing the full exploitation of latency in large digital networks. Although the basic method could be generalized for any type of circuit, thus far the scope has been limited to MOS integrated circuits.

A MOS integrated circuit can be regarded, for simulation purposes, as a nonliniear time-invariant network, whose components are linear resistors, nonlinear capacitors, nonlinear controlled current sources and independent voltage sources. Methods for writing a set of equations that charactarize this type of network are well established. If a nodal formulation is used, the equation for a MOS network has the general form shown in equation (1), where v is a vector representing the node voltages, C(v) is a matrix whose elements represent the nonlinear capacitors, and f() is a vector that represents the currents through all other elements.

$$C(v)\frac{dv}{dt} + f(v) = 0 \qquad (1)$$

To solve this set of first-order nonlinear differential equations, for which an initial solution is assumed, most techniques use difference methods: the interval over which the solution for (1) is to be evaluated, is divided into small intervals, corresponding to time steps. Let h represent a time step and t_k and $t_{k+1}=t_k+h$ two consecutive points on the time axis for which a solution of (1) is to be found. If the value of v_k is known, the value of v_{k+1} can be evaluated by discretizing the derivative operator: the derivative dv/dt can be replaced, for example by $(v_{k+1}-v_k)/h$. If C(v) and f(v) are estimated using only known values of v, (for example v_k) an explicit integration method is obtained and v_{k+1} can be immediately computed. Unfortunately, explicit methods tend to be unstable if the time step is large compared to the smallest time constant in the circuit. Therefore, most circuit simulators use implicit methods where the value of C and f is assumed to be some function of v_{k+1}. The new voltage values v_{k+1} can be evaluated

by solving a set of nonlinear equations. This is done in most cases by some variation of Newton's method, where the nonlinear elements are replaced at every iteration by a linear approximation evaluated for the estimate of v_{k+1}. To summarize, conventional methods first replace the derivatives by some estimate and only then linearize the network to solve the resulting system of nonlinear equation.

As motivation for the method developed here, we observe that to solve a nonlinear circuit, it is natural for the step size to be controlled by the region over which a linearized approximation to the device behavior is valid. This is particularly interesting for MOS networks, where for large ranges of the voltages the device behavior is quasi-linear. If the elements in the network are linearized the dynamic behavior of the network can be represented by a system of linear differential equations:

$$C\frac{dv}{dt} + Gv = i \tag{2}$$

In this equation C and G are n by n matrices, and v and i are n-vectors where n is the number of nodes in the network. For a MOS circuit, C is a symmetric, nonsingular matrix and in most cases diagonally dominant. While diagonal dominance is not essential to the algorithm, it can be used to speed the solution of (2).

Equation (2) is a first order differential equation, that could in principle be solved by any of the well established integration methods, all of which are computationally intensive. The analytical solution to (2) can be easily formulated as a function of the matrix e^{AT}, and where $A=C^{-1}G$. Although this matrix is in general hard to compute, if the order of the matrix is small it is easy to obtain an explicit and exact solution for v.

Note that equation (2) will be valid only for the values of v where the linear model in use is valid. Whenever a node voltage attains a value on the present limit of its region of linearity, the integration should be halted, and new linear approximations for the nonlinear elements reevaluated. We will refer to the points on the time axis where these limits are attained as events for the node. We are thus led to an algorithm that will compute for every node the associated event, and accept the integration up to the time point that corresponds to the next event.

A straightforward method to implement such an algorithm is to assume that the voltage values are multiples of ΔV. This discretization ultimately defines the accuracy of the simulation. At every step the objective of the integration algorithm is to find, for each node in the circuit whose voltage is changing, the point on the time axis where the node voltage attains the next discrete value (an event). Note that our technique differs from traditional integration schemes in that we do not try to compute (all) circuit variables for certain points on the time axis, but rather we will have events (points on the time axis) for (all) nodes in the circuit, corresponding to discretized values of the voltages.

Furthermore, the algorithm is clearly explicit thereby avoiding costly iterations at each time step. As bounds for the voltage values are automatically maintained, the method does not suffer stability problems that occur in conventional explicit techniques due to wild voltage variations when the step size is not kept small.

A program, called CINNAMON has been developed to implement this approach. It has thus far proven to be quite reliable at significantly reduced computational costs.

4. WASIM: A Waveform Based Simulator for VLSICs

A new approach for simulation for cell-based digital MOS designs has been developed which is aimed at achieving circuit level accuracy with logic level speed. This approach exploits the regularity which is typical of large VLSI designs by building macromodels of individual cells. These macromodels are generated from parameterized analytical models which charaterize the waveforms produced by typical device-level output stage topologies. Inputs of these macromodels are designable parameters such as layout, as well as circuit parameters such as fan-out. An event-driven, waveform-based simulator, called WASIM has been developed which employs these macromodelled cells.

Specifically, we have developed a method for accurately modelling analog waveforms produced by typical cells, and developed efficient models which relate these waveforms to input waveforms, loading conditions, and cell parameters such as device dimensions. An event driven simulation algorithm has also been developed that uses such models, handles feedback loops in a natural way, and which does not require backtracking in time. We have implemented this event-driven simulation algorithm in a program called WASIM as well as a software package to automatically generate cell macromodels for WASIM.

5. The Use of Hardware Accelerators for Algorighmic Level Descriptions

The problem being studied in this part of the research is how to improve the performance of simulators for digital systems described at the algorithmic level. The method chosen to reduce the execution time of simulation is to use a logic simulation engine to simulate a data flow representation. The data flow representation is derived from an algorithmic description of a digital system. This section will discuss why this method was chosen and the specific data flow representation being used. The method used to convert between the algorithmic description and the description for the logic simulation engine will be given. To check the method a set of experiments will be described which estimate the performance of the simulator running on a logic simulation engine. Finally the results of the experiments and a summary will be given.

To use a logic simulation engine the simulation proceeds in the following way:

1. The designer writes description of digital system's behavior in ISPS a hardware description

language,

2. The ISPS description is automatically translated into a data flow graph called the Value Trace which exhibits the same behavior as the ISPS,

3. The Value Trace is converted into a gate network which can be simulated on a logic simulation engine,

4. The logic simulation engine's features can be used to examine values which can be mapped back to the values in the ISPS description allowing the designer to check the design.

The first two steps are already part of the Carnegie-Mellon Design Automation (CMU-DA) system[1]. The first step has been described elsewhere and will not described here. The reason for doing the second step is described next.

5.1. Data Flow Representation

The ISPS description is converted into a data flow representation before being converted into a gate network for several reasons. First, data flow graphs map nicely onto a event driven simulation because of the sparseness of activity in the graph. Also data flow graphs eliminate unnecessary temporary storage elements which reduces bus traffic in logic simulation engines. This is true for logic simulation engines which use different processors to model gate elements and storage elements. Logic simulation engines which use separate processors include the IBM YSE, ZYCAD Logic Evaluator and NEC HAL[2].

The data flow representation used in this research is the Value Trace (VT)[3]. The Value Trace is a directed acyclic graph where the nodes represent operators and the arcs represent data dependencies between the operators. The structure is much like the data flow graphs used by optimizing compilers.

The Value Trace is automatically generated from an ISPS description. Figure 5-1 shows some ISPS code on the left and the corresponding Value Trace on the right. Each of the boxes on the right represents a Value Trace operation. The number in the upper left-hand corner of each box is the operator number. The operator type is described in the center of each box. The inputs to the operator are on the top and the outputs on the bottom. Conceptually, an operator does not produce an output value until all its inputs have valid values. In this way values pass down the lines in the diagram.

The open boxes at the top of the figure represent inputs to the data flow graph. The open boxes at the bottom of the figure represent outputs from the data flow graph. The data flow graph with its inputs and outputs is called a *vt-body*. A vt-body corresponds to a labeled block, loop or procedure in the ISPS description.

Operations are divided into data, arithmetic and logic and control operators. An example of a data

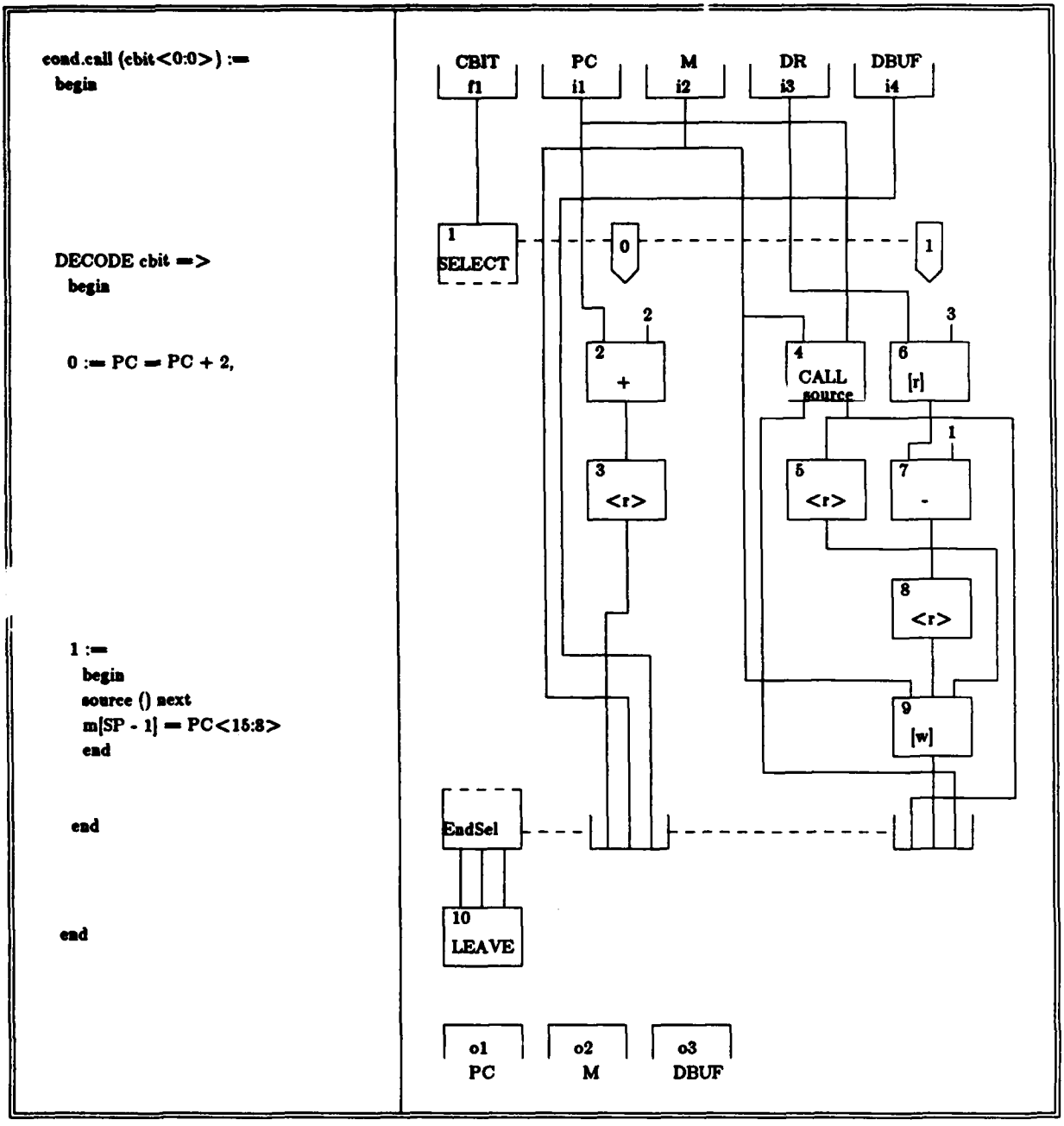


Figure 5-1: ISPS and Value Trace Example

operator is operator number nine which writes a value to an array element. An example of an arithmetic operator is operator number two which is a `plus` operator. The operator has two inputs one comes from the vt-body input PC and the other is a constant of value two. When all the inputs to the operator are valid (constant inputs are always valid) the operator produces the output value PC + 2. Other data operators are provided to read and write array and single values.

Several control operators are provided in the Value Trace including subroutine call, select between different actions and looping. Operator number one is a `select` operation. When its input becomes

valid the branch with the corresponding value (the value is in the pentagon above the branch) is activated. For example if CBIT has the value 0 then the first branch is activated and the operations in the branch (operators number 2-3) produce outputs when their inputs are valid. The operators in the other branch (operators number 4 - 9) will not produce outputs even if their inputs are valid. The output values of the select operator come from whichever branch is activated.

Another control operator in the figure is operator four, a subroutine `call`. When all the inputs to the call are valid the call operator copies its input values to the start of another vt-body in this case named `source`. When all the values have traversed the other graph they will be passed back through the outputs of the calling operation.

5.2. Conversion to Gate Network

To simulate the Value Trace the VT is translated into a gate network which can be simulated on a logic simulation engine. The conversion between Value Trace and gate network is done by replacing each of the operators with its gate equivalent.

This is relatively simple for the arithmetic and logic operators such as `plus`, `minus`, `and` and `or`. Figure 5-2 shows how a plus operator is converted to gates. In the figure the Value Trace plus operator which is two bits wide is converted to two exclusive-or gates to generate the sum and two majority gates to generate the carry. These models produce values continuously when their inputs change their outputs change.

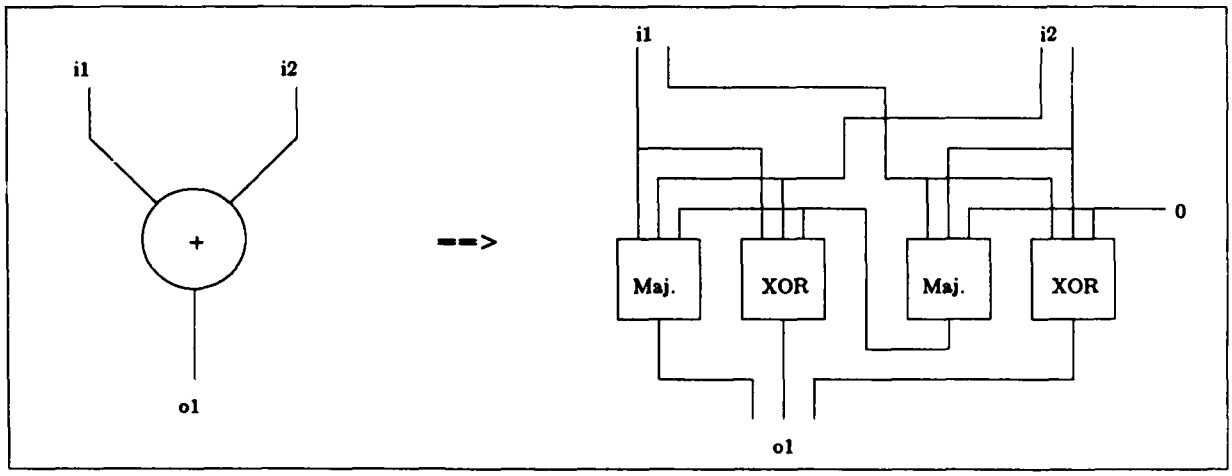


Figure 5-2: Conversion of Arithmetic Operator

Data operators that change the values of storage elements must wait for all their input to be valid otherwise random locations and/or values would get written. As an example if the data for an array read arrived before the address the data would get written into whatever location was last addressed. The

method used to prevent random writes is discussed next along with control operators.

The conversion to gates is more complex for the control operators such as selects of different branches and calls of subroutines. These operators like some of the data operators must wait until all their inputs are valid.

To prevent control operators and storage accesses from occurring before they should the following "one hot" control scheme was adopted. A token is passed along with the slowest data to trigger the control operators, array reads, array writes and register writes. A control scheme like this one was used in the Digital Equipment Corporation PDP-16[4].

Figure 5-3 shows a simplified version of the Value Trace from Figure 5-1 with the paths for the control tokens added. The token moves down the figure along the heavy lines. When the token comes to a `select` operation the token is only allowed down the branch whose activation value matches the value at the input to the select. So if the value 1 is the input to the `select`, the second branch over will be activated allowing the token to activate the array write operator (number 9) symbolized by the box labeled `[w]`.

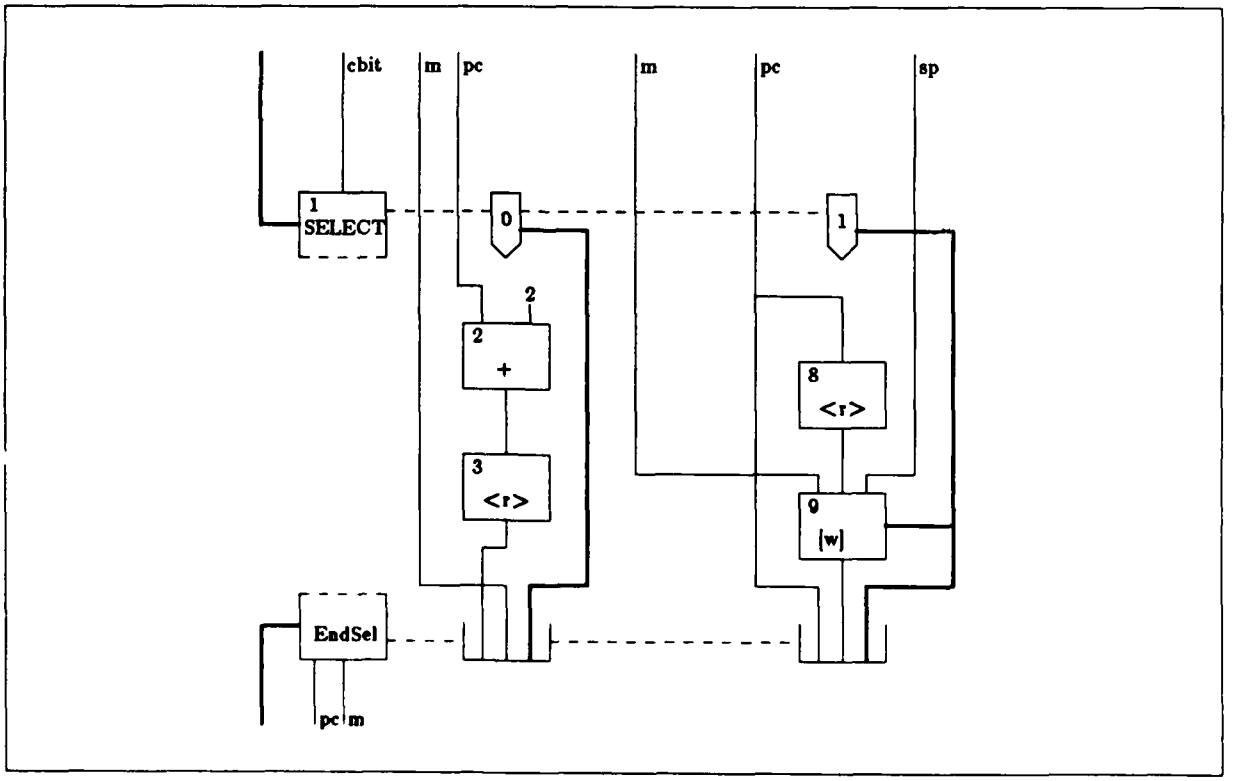


Figure 5-3: Example of Control Operator

Along with the control scheme some extra registers have to be added to the gate network to hold values for control operators These operators include subroutine returns and restarts of loops. Storage is required on subroutine returns because each call could produce a different value. The returned values are stored at the point where the subgraph was called. Looping operators have storage to break up loops and prevent race conditions.

After all the operators have been converted in gates and registers the gate network can be used to simulate the behavior of the data flow description. The gate network used for the data flow simulation is not the same as the gate network that would be used to implement the actual system. The networks are not the same because they have a different set of aims. The gate network for simulation has been designed to be easy to generate from an algorithmic description and to make it easy to evaluate with an event driven logic simulation engine. The actual design for the system, which has not been completed at this point in the design process, will try to make efficient use of the gates.

Efficient use of gates leads to more gates being active in any one time step and also a large percentage of the gates being active in any one time step. Both these facts will decrease the performance of event driven simulators. The size figures for the simulation networks will be given in the next section.

5.3. Preliminary Results

In this section three simulators are compared. The first is the ISPS simulator which is used as an example of a conventional algorithmic level simulator. The second is the software VT simulator which was used to estimate the performance of the third simulator, the VT simulator, running on a logic simulation engine. Results are given for the third simulator running on a ZYCAD Logic Evaluator and a IBM Yorktown Simulation Engine.

Simulators Compared

Figure 5-4 shows that all three simulators will use the same ISPS description as input. In all three simulators the ISPS is converted into a parse tree called the GDB by the ISPS parser. To use the ISPS simulator the GDB is converted into RTM code. The Register Transfer Machine (RTM) is an imaginary computer which is implemented in software and executes the RTM code to simulate the ISPS description. The Register Transfer Machine interprets the RTM code.

To use the software VT simulator the GDB is converted into the Value Trace data flow representation by the VT translator. C code is produced from the Value Trace which models the operators and networks in the Value Trace. These routines are combined with some simulation routines to form the software VT simulator. This simulator is a compiled software simulator.

The hardware VT simulator will be used much like the software simulator only instead of the Value

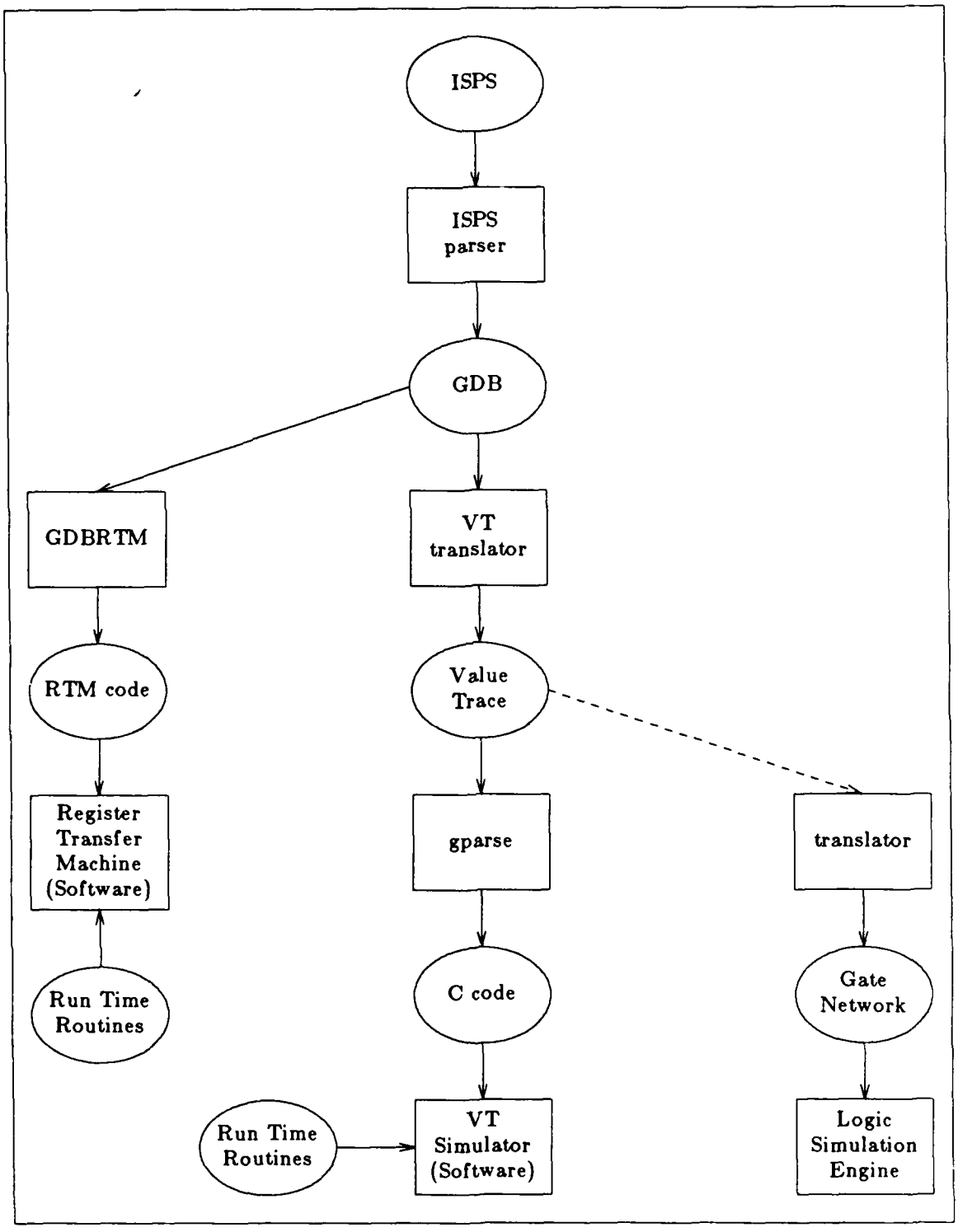


Figure 5-4: The Three Simulators Compared

Trace being translated into C code it will be translated into a gate network as described above. This network, when evaluated, will simulate the Value Trace formed from the ISPS description.

Measures Used

The software VT simulator was used to estimate the performance of the VT simulator running on the logic simulation engines. Both static and dynamic measures were used to evaluate the performance of the simulator. The static measures depend only on the design being simulated, while dynamic measures depend on the data being run through the simulated system.

Static measures include the size of the gate network and the number of storage elements needed to model the Value Trace. Dynamic measures include the number of elements evaluated and the number of storage elements accessed when simulating the Value Trace. Gate depth is a measure of how many levels of gating are needed to implement the one pass through the Value Trace.

Estimates of the number of gate activations, register writes and memory reads and writes were collected using the software simulator. When an input to an operator changes, the number of gates in the operator is added to the total number of gate activations. The number of gate activations is updated on every input change to model the ZYCAD Logic Evaluators event driven algorithm. The number of register and array activations is only updated after all the inputs are valid. This is done because these operation are only activated after the control token arrives.

Discussion

The tables in this section give the results for simulating a MOS 6502 microprocessor doing a double precision multiply. Several other programs have been run which yield similar results including a *Tower of Hanoi* program written in C. The values have been calculated for two conditions one where all the gates are counted and the other where the gates used for control are not counted.

Table 5-1 gives the static data counts for the MOS 6502. The first column gives the number of gates used to model the whole 6502 and just the data part of the 6502. The second column gives the number of bits of register needed to model the 6502 followed by the number of separate registers in the network.

Double Precision Multiply	Static Measures	
	Network Size	Register Storage
With Control	25400	1829/70
Without Control	1063	93/13

Table 5-1: Static Data from Software Simulator

The table shows that most of the gates are being used to do control. Only four percent are used to do

data operations. This finding will be discussed latter.

Table 5-2 gives the dynamic data counts for the 6502 running the multiply program. Values are given for executing the whole program and on a per instruction basis. The per instruction numbers come from dividing the total number of gate activations by the number of 6502 instructions executed.

Double Precision Multiply		Dynamic Measures				
		Gate Act.	Memory Reads	Memory Writes	Register Writes	Gate Depth
With Control	Total	1041595	835	116	1573	10718
	Average	4268	3	0.5	6	43
Without Control	Total	30672	835	116	790	–
	Average	125	3	0.5	3	–

Table 5-2: Dynamic Data from Software Simulator

The number of gate activations is shown in the first column. As with the static counts most of the gate activations are being used to do control. The second, third and fourth columns give the number of register and memory accesses. The number of register and memory reads and writes give some measure of how often data must be transferred to and from the ZYCAD or YSE processor doing the storage operations.

The gate depth number can be used to estimate the amount of parallelism in the gate network. Since the average instruction takes 4268 gate evaluation in a network 43 gates deep each level will have approximately 100 gates in it. This number is somewhat optimistic because of the difference between the way the Value Trace simulator and the ZYCAD evaluate arithmetic operations. As an example, an add instruction will be simulated by the VT simulator in one time step, while the ZYCAD simulator will take multiple time steps because of the carry propagation.

Table 5-3 gives a comparison of the three simulators and the actual microprocessor. The first line was calculated from the instruction executed using the figures published by the manufacturer of the 6502 microprocessor. The second and third lines are the CPU times used by the simulators running on a VAX™ 11/785 under UNIX™. The fourth line and fifth lines are the estimated performances for the hardware VT simulator. The sixth line is the performance of the N.mPC simulator which uses a software method to get higher performance than the ISPS simulator.

To get the upper bound on the ZYCAD simulation time it was assumed there was no parallelism in the design so all the elements had to be evaluated sequentially. This number comes from dividing the number

Simulator	Machine	Speed	Slow Down
(Real)	6502	976 μsec	1
ISPS	VAX 11/785	15 sec	15,000
VT Software	VAX 11/785	188 sec	200,000
VT Hardware	ZYCAD	1 sec - 65 msec (est)	1000 - 65 (est)
VT Hardware	YSE	90 msec (est)	90 (est)
N.mPC	PDP 11/70	n/a	2000

Table 5-3: Speed Comparisons of Simulators

of element evaluations needed by the element evaluation rate of one ZYCAD *event processor*. The lower bound on the simulation time assumes that there is enough parallelism in the design to keep all the event processors busy all the time. This value is equal to the upper bound divided by the number of event processors (sixteen in the largest ZYCAD Logic Evaluator). These numbers ignore the time needed to get stored values. This was done because no figures were available for these times.

The YSE evaluates every gate on every time step. An estimate of the number of gates it has to evaluate is the total number of gates in the network times the gate depth (the number of time steps). This number can then be divided by the YSE evaluation speed to get the run time of the simulator.

5.4. Summary

This chapter has described work to improve the performance of algorithmic level simulators. The approach chosen was to convert the algorithmic description into a dataflow representation and then into a gate network. The gate network will be simulated using a logic simulation engine.

The method for converting to gates was discussed and what extra control has to be added to gate network to allow it to model a data flow description. A software simulator was used to estimate the performance of the simulator running on a logic simulation engine.

The estimates were calculated for two different logic simulation engines one which uses a event driven and one that uses a compiled simulation algorithm. The preliminary results show a speed up of between 16 and 230 over a conventional algorithmic simulator. It is hypothesized that this method works best for event driven logic simulation engines.

Some problems in the method were brought out. The most important problem is the large number of gates needed to model control operators. As part of our future research, we will be running these models on the YSE, ZYCAD, and possibly the MIT data flow simulator to obtain actual results. It is expected that these results will lead the design of a hardware simulation engine for the algorithmic level of design.

6. Publications and Student Support

Over the course of the contract, five students have been supported. These are listed in Table 6-1. The publications generated by this research contract are listed in Table 6-2.

Table 6-1: Personnel Supported

Katherine Hsu	M.S. degree, graduated 1984
Sani Nassif	Ph.D. degree, graduated 1986
John Beardslee	M.S. degree, graduated 1986
Robert Blackburn	M.S. degree, graduated 1985
David Geiger	Ph.D. candidate

Table 6-2: Publications

1. Thomas, D.E., Hitchcock III, C.Y., Kowalski, T.J., Rajan, J.V. and Walker, R.A., "Methods of Automatic Data Path Synthesis", IEEE Computer Society's Computer Magazine, Invited paper. December, 1983.

2. Nassif, S.R., Strojwas, A.J., and Director, S.W., "FABRICS II -- USER'S MANUAL, A Program for the Statistical Simulation of the IC Fabrication Process", SRC Research Report No. CMUCAD-83-15, October, 1983.

3. Strojwas, A.J., Nassif, S.R., and Director, S.W., "Optimal Design of VLSI Minicells Using a Statistical Process Simulator", Proceedings of ISCAS, April, 1983.

4. Thomas, D.E. and Nestor, J.A., "Defining and Implementing a Multi-Level Design Representation with Simulation Applications", IEEE Transactions on Computer-Aided Design of Circuits and Systems, July, 1983.

5. "Linking the Behavioral and Structural Domains of Representation in a Synthesis System", Blackburn, R.L. and Thomas, D.E., 22nd ACM/IEEE DAC Conference, Las Vegas, NV, June 23-26, 1985.

6. "WASIM: A Waveform Based Simulator for VLSICs", S. R. Nassif, and S. W. Director, Proceedings of 1985 International Conference on Computer-Aided Design, pp. 29-31, November 1985.

7. "A Macromodelling Approach for Simulation of VLSI Circuits", S. Nassif, Ph.D. Dissertation, completed May 1986, CMUCAD-86-15.

8. "CINNAMON: Coupled Integration and Nodal Analysis of MOS Circuits", S. W. Director, L. Vidigal, and S. R. Nassif, Proceedings of the 23rd Design Automation Conference, pp. 179-185, June 1986.

9. "Linking the Behavioral and Structural Domains of Representation for Digital Systems Design" D.E. Thomas, R.L. Blackburn, J.V. Rajan. IEEE Transactions on Computer-Aided of Circuits and Systems. To be published in January, 1987.

10. "Delilah 3: An Improved User-Machine Interface", Katherine Hsu, MS Thesis, CMUCAD-84-25, February 1984.

11. "Implementation of a Logic Simulator and Mixed Level Simulation for SAMSON2", John Mark Beardslee, MS Thesis, CMUCAD-86-12, May 1986.

12. "Linking Behavioral Representation in an IC Design Systems", Robert L. Blackburn, MS Thesis, CMUCAD-84-31, May 1984.

References

1. Donald E. Thomas, Charles Y. Hitchcock III, Thaddeus J. Kowalski, Jayanth V. Rajan, Robert Walker, "Automatic Data Path Synthesis", *Computer*, Vol. 16, No. 12, December 1983, pp. 59-70.

2. Nobuhiko Koike, Kenji Ohmori, Tohru Sasaki, "HAL: A High-Speed Logic Simulation Machine", *IEEE Design and Test of Computers*, Vol. 2, No. 5, October 1985, pp. 61-73.

3. M. McFarland, "The VT: A Database for Automated Digital Design", Tech. report, Carnegie-Mellon University, December 1978.

4. C. Gordon Bell, John Grason, Allen Newell, *Designing Computers and Digital Systems*, Digital Press, 1972.

Table of Contents

List of Figures

List of Tables

END

2-87

DTIC